Monte Carlo Solution of Bond Percolation Processes in Various Crystal Lattices

By H. L. FRISCH, S. B. GORDON, V. A. VYSSOTSKY, and J. M. HAMMERSLEY†

(Manuscript received November 29, 1961)

We present the outline of an IBM 7090 machine program for the Monte Carlo estimation of the percolation probability for a variety of space lattices. The underlying theory is briefly summarized.

I. STATEMENT OF THE PROBLEM

Percolation processes deal with the transmission of a "fluid" (disturbance, signal, etc.) through a "medium" (material, region, etc.) against impediment by random irregularities situated in the medium.[1] This paper considers the case where the medium is a regular crystal lattice in two or three dimensions, consisting of "atoms" (the vertices or sites of the lattice) and "bonds" joining specified pairs of atoms. The next section will specify the structure of the lattice more completely. The fluid originates at one or more atoms of the lattice, called the source atoms, and flows from atom to atom along the connecting bonds. However, each bond (independently of all other bonds) has a fixed probability p of being able to transmit fluid and a probability $q = 1 - p$ of being blocked: these randomly situated blocked bonds constitute the random impediments to the spread of the fluid. We write $P_N(p)$ for the probability that the fluid will reach (or "wet", as we shall say) more than N other atoms besides the source atoms; and the problem is to estimate $P(p) = \lim_{N\to\infty} P_N(p)$. We do this by estimating $P_N(p)$ for a suitably large value of N: it turns out that $N \sim 2000$ is sufficient in many cases. The present paper describes the general organization of an IBM 7090 program for obtaining a Monte Carlo estimate of $P_N(p)$. The numerical results appear elsewhere.[2]

† Oxford University, Oxford, England.

II. STRUCTURE OF THE LATTICE

We consider below the three-dimensional problem in such a way that it contains as a special case the two-dimensional problem. A regular three-dimensional lattice consists of a number of fundamental cells, all identical apart from their position in space. Each cell is specified by integer coordinates (x, y, z) representing its position in space ($x, y, z = 0, \pm 1, \pm 2, \cdots$). Each cell contains a finite number of atoms, limited in our IBM 7090 program to a maximum of eight atoms per cell, and denoted by $A, B, \cdots, H$.

In the problem, as originally posed in the previous section, some of the bonds may be one-way (i.e., only able to transmit fluid in a specified direction) while others may be two-way (i.e., able to transmit fluid in either direction). It can, however, be proved theoretically that $P_N(p)$ is unaltered if a two-way bond is replaced by two one-way bonds of opposite directions. It is therefore enough to consider the case where all bonds are one-way bonds, and we confine ourselves to this case hereafter.

Let us write $T(x, y, z)$ for the atom of type $T (T = A, B, \cdots, H)$ in cell (x, y, z). The lattice is regular in the sense that, if there is a bond from $T(x, y, z)$ to $T^*(x^*, y^*, z^*)$, then there is a bond from $T(x + l, y + m, z + n)$ to $T^*(x^* + l, y^* + m, z^* + n)$ for any $l, m, n = 0, \pm 1, \pm 2, \cdots$. Therefore we need only specify the terminal atoms reached by each bond from each atom of the cell (0, 0, 0). The program limits the number of bonds from a given atom to a maximum of 12. There is no restriction that distinct bonds from a given atom shall lead to distinct terminal atoms; and thus we may, if we wish, have two or more bonds in the same direction between a given pair of atoms.

The machine receives information about the lattice structure from a series of input cards, having the format described below. To each atom of the cell (0, 0, 0) there is a "structure" card, followed perhaps by one or two "continuation" cards. The format of a structure card is:

Columns	5–13	15–23	24–26	27–35	36	37–45	46	47–55	56	57–65	70–72
Contents	*STRUCTURE*	$T(+0+0+0)$	$T\emptyset\cdot$	$T(xyz)$	,	$T(xyz)$	,	$T(xyz)$	,	$T(xyz)$	*CT*1

In such a card, T (in columns 15, 27, 37, 47, 57) stands for one of the letters $A, B, \cdots, H$ (not necessarily the same in each case); and each of the symbols x, y, z is a *signed* integer. The entries in columns 5–13, 24–26, 36, 46, 56, 70–72 are in BCD (binary coded decimal). Instead of the whole word *STRUCTURE* in columns 5–13, the single letter *S* in column 5 will suffice. Columns 15–23 specify the atom of cell (0, 0, 0) from which bonds lead to the atoms appearing in columns 27–35, 37–45, 47–55, and 57–65 respectively. Thus the structure and card provide for

up to four bonds from the given atom of cell (0, 0, 0). If there are fewer than four bonds, some of the $T(xyz)$ will be left blank. The next four (or fewer) bonds are similarly specified on a first continuation card with format:

Columns	5–7	27–65	70–72
Contents	*CT*1	Same format as structure card	*CT*2

The last four (or fewer) bonds appear on a second continuation card with format:

Columns	5–7	27–65	70
Contents	*CT*2	Same format as structure card	*S*

In both continuation cards, columns 5–7 and 70–72 are in BCD. If the atom specified in columns 15–23 of the structure card has four or fewer bonds from it, both continuation cards are omitted and we replace CT1 in columns 70–72 by *S* in column 70 (in BCD); if it has between 5 and 8 (both inclusive) bonds from it, the second continuation card is omitted and we replace CT2 in columns 70–72 by *S* in column 70. The final structure on continuation card is followed by an "end" card with *END* (in BCD) in columns 5–7.

For example, the simple cubic lattice with a pair of one-way bonds (one in each direction) between each pair of nearest-neighbor atoms is specified by:

```
STRUCTURE  A(+0+0+0)  TO-  A(+1+0+0),  A(+0+1+0),  A(+0+0+1),  A(−1+0+0)  CT1
CT1        A(+0+0+0)  TO-  A(+0−1+0),  A(+0+0−1)                           S
END
```

Similarly, the tetrahedral lattice (diamond crystal) with a pair of one-way bonds in each direction between nearest neighbors is given by:

```
STRUCTURE  A(+0+0+0)  TO-  B(+0+0+0),  B(+1+0+0),  B(+0+1+0),  B(+0+0+1)  S
S          B(+0+0+0)  TO-  A(+0+0+0),  A(−1+0+0),  A(+0−1+0),  A(+0+0−1)  S
END
```

The two-dimensional cases arise when $z = 0$ identically on all cards.

III. INPUT TO THE MACHINE

The complete input to the machine consists of the program deck, followed by (*i*) an identification card, followed by (*ii*) a parameter data card, followed by (*iii*) a set of structure and continuation cards (as described above), followed by (*iv*) an end card (as described above).

The format of the identification card is:

Columns	5–15	18–25	26–66	68–72
Contents	*PERCOLATION*	*CRYSTAL*	Name of crystal	*PUNCH*

These entries are all in BCD. If *PUNCH* appears in columns 68–72, the output from the machine will appear on the printout and will also be punched onto cards; if *PUNCH* is omitted from columns 68–72, the output will appear on the printout but cards will not be punched.

The format of the parameter data card is:

Columns	5–8	10–12	13	14–16	19–23	25–28	31–37	39–47	48	49–57
Contents	*RUNS*	***	–	***	*LINES*	****	*SOURCES*	$T(xyz)$	,	$T(xyz)$

Columns 5–8, 13, 19–23, 31–37, 48 are in BCD. Columns 10–12 contain a first run number; columns 14–16 contain a last run number: the purpose of these two run numbers will be described presently. All run numbers must be *positive* integers. Columns 25–28 contain the number N, appearing as a suffix in the desired function $P_N(p)$. The maximum value of N permitted by the program is 7000.

IV. GENERAL CONDUCT OF THE CALCULATION

The calculation carried out by the machine is at first sight rather different from the problem posed in the first section of this paper. The change of formulation simplifies the calculation without affecting the ultimate numerical answer.

The calculation consists of a number of separate runs. In each individual run, independent random numbers η are assigned to each bond of the lattice. Each η is uniformly distributed between 0 and 1. This process of assigning η-values to the bonds replaces the original process of blocking bonds, so that the question of a bond being blocked or not does not arise in the reformulated process. Consider a connected path of bonds $\beta_1, \beta_2, \cdots, \beta_k$ on the lattice, where the (necessarily one-way) bond β_i leads *from* the atom *to* which the preceding bond β_{i-1} led $(i > 1)$. Let $\eta_1, \eta_2, \cdots, \eta_k$ be the η-values of the respective bonds $\beta_1, \beta_2, \cdots, \beta_k$. Define the η-value of this path to be the minimum of $\eta_1, \eta_2, \cdots, \eta_k$. Next consider any atom other than a source atom. Define the γ-value of this atom to be the supremum of all path η-values, taken over all paths which lead from some source atom to the given atom. Finally define c_n to be the largest number such that there are more than n atoms, other than source atoms, whose γ-values are equal to or greater than c_n.

Now c_n for any fixed n is clearly a random variable, depending upon the several η-values assigned to bonds of the lattice. $P_n(p)$, regarded as a function of p, is the cumulative distribution function of the random variable $1 - c_n$.[3]

The machine is programmed to calculate c_n for each run. Thus the set of all runs provides a sample of values of $1 - c_n$, and the empiric

distribution of this sample can be taken as an estimate of the required function $P_n(p)$.

So far we have regarded n as a fixed integer. Actually, the output of the machine on any one run is a table of c_n as a function of n for all $n \leqq N$, where N is the number set on the parameter data card. Thus the complete calculation provides estimates for $P_n(p)$ for all $n = 1, 2, \cdots, N$ and all $0 \leqq p \leqq 1$.

Successive runs are numbered serially $R_1, R_1 + 1, \cdots, R_2$, where R_1 is the first run number specified on the parameter data card, and R_2 is the final run number specified on the parameter data card. Hence $R_2 - R_1 + 1$ is the sample size for each empiric distribution. The run number $R_1 + i$ of the $(i + 1)$th run is used to trigger off the generation of random numbers η assigned to bonds in this run. Hence a run may be repeated for checking purposes by repeating the run number; but, if a fresh and independent sample of γ_n is desired, the parameter data card must specify a set of run numbers which does not overlap the set previously used. Since three decimal digits are available for run numbers, the maximum sample size is 999. A sample of size about 100 is usually adequate.

V. OUTPUT FROM THE MACHINE

The printout from the machine, also punched onto cards if ordered on the identification card during input, is as follows.

The printout begins with a copy of all input data (excluding the program deck). Thereafter follows a table, whose columns are headed:

RUN NØ. STATE c VALUE

Each row of the table has an entry in each of these four columns. An entry under *RUN* is the current run number R being computed ($R_1 \leqq R \leqq R_2$). The entries under *NØ.* and *c VALUE* are respectively n and c_n, tabulated for $0 \leqq n \leqq N$ omitting any values of n such that $c_n = c_{n-1}$ $(0 \leqq n \leqq N)$. Thus the only values of c_n printed are new values less than all preceding values in the run. (Clearly c_n is a non-increasing function n by virtue of its definition.) Such new values are indicated by the prefix *NEW*. The value of c_N is, on the other hand, prefixed by *FINAL* and also suffixed by an asterisk to help in reading the output quickly. The entry under *STATE* is either *FINISHED* or *TØ BE CØNT.*, according to whether the run is complete or not. The only other possible entry under *STATE* is *INHIBITED*: this is a safety device, to be explained later. It means that certain technical circumstances

TABLE I — EXTRACT OF PRINTOUT

RUN	*NØ.*	*STATE*	*c VALUE*
23	518	*TØ BE CØNT.*	*NEW* $c = .748$
23	1000	*FINISHED*	*FINAL* $c = .748*$
24	0	*TØ BE CØNT.*	*NEW* $c = .906$
24	1	*TØ BE CØNT.*	*NEW* $c = .803$
24	5	*TØ BE CØNT.*	*NEW* $c = .791$
24	11	*TØ BE CØNT.*	*NEW* $c = .783$
24	17	*TØ BE CØNT.*	*NEW* $c = .770$
24	19	*TØ BE CØNT.*	*NEW* $c = .762$
24	20	*TØ BE CØNT.*	*NEW* $c = .715$
24	21	*TØ BE CØNT.*	*NEW* $c = .711$
24	1000	*FINISHED*	*FINAL* $c = .711*$

(whose occurrence is extremely unlikely) have arisen to prevent completion of the run. In an inhibited run all values of c_n in the printout are valid: all that has happened is that n has been prevented from rising beyond a certain value, at which instant the run is automatically discontinued.

The extract of the printout shown as Table I will help clarify matters. It gives the end of the twenty-third and the whole of the twenty-fourth run for a computation on the simple cubic crystal with $N = 1000$. To find the value of c_n for a value of n not printed in the Table, take the value of c for the largest n less than the required n. In the above example, $c_8 = 0.791$ in run 24.

The machine stores values of c as 9-bit binary decimals. Hence the rounding error in c is about 0.001 and not 0.0005.

VI. OUTLINE OF THE PROGRAM

What has been said so far contains everything that a user of this program needs to know. What follows in the remainder of this paper is an explanation of how the machine carries out the program, and is intended for those who are interested in programming techniques.

The two main entities in any given run are (*i*) a number denoted by c, and (*ii*) a so-called "wet list" of atoms. An atom is qualified for membership on the wet list if it is a source atom or if its γ-value is not less than c. Normally the value of c is held constant and the machine recruits new members of the wet list. However, if a stage is ever reached during a run where no further recruits can be found with the existing value of c, then the machine reduces c by an amount just sufficient to ensure the existence of at least one fresh recruit. The run begins with $c = 1$ and a wet list containing just the source atoms. Since a given

atom in a given run has a given γ-value (depending upon the η-values assigned to bonds in that run), reduction of c can never disqualify existing membership of the wet list. Successively recruited members of the wet list (other than source atoms) are numbered 0, 1, $\cdots$. A little reflection will show that at the moment, when the member numbered n is added to the wet list, the current value of c must be c_n. The run is terminated as soon as the wet list contains $N + 1$ members. Thus the machine output is simply the result of printing any freshly reduced value of c against the number of the next member to be added to the wet list.

This procedure would be unworkable if the machine had to examine all atoms of the lattice for this qualification to belong to the wet list. What makes the procedure workable is the observation that there can exist no qualified fresh recruits if the existing value of c exceeds the η-values of all bonds, which lead from some atom of the wet list to some atom not in the wet list. Let us call such bonds the *outgoing* bonds of the wet list. If the wet list has an outgoing bond whose η-value is at least c, then this bond leads to an atom which is qualified for membership of the wet list. If the value of c has to be reduced, the new value of c is the highest η-value of all existing outgoing bonds. Hence, at any stage of the calculation the machine need only examine the η-values of the outgoing bonds of the currently existing wet list.

Information about the current status of the wet list resides in a block of registers in the core storage of the machine, with three registers (denoted by E, $E + 1$, $E + 2$) allocated to each atom of the list. We write

$$\begin{aligned} E &= (x, y, z, T) = [11, 11, 11, 3]; \\ E + 1 &= (X) = [36]; \\ E + 2 &= (\psi, \phi, \pi) = [9, 12, 15]; \end{aligned} \tag{1}$$

to indicate that E contains four different quantities (denoted by x, y, z, and T respectively) occupying 11, 11, 11, and 3 bits of the 36 bits available in a single register. The contents of $E + 1$ and $E + 2$ are exhibited in a similar manner. If we wish to emphasize that we are talking about the nth atom A_n ($n = 0, 1, \cdots, N$) in the wet list (excluding the source atoms of the wet list), we place n as a suffix to any of the above quantities: thus

$$(E + 2)_n = (\psi_n, \phi_n, \pi_n). \tag{2}$$

The symbols have the following meanings: (x_n, y_n, z_n) are the three integer coordinates of the cell containing the atom A_n, and T_n is the type of the atom A_n ($T_n = A, B, \cdots, H$ in the notation for crystal structure). X_n is a 36-bit pseudo-random number, generated according to the multiplicative congruential recursive relation (low multiplication)

$$X_n \equiv \beta^4 X_{n-1} \pmod{2^{36}}, \tag{3}$$

where β is a permanently fixed odd integer, and the recursion is triggered from the initial state

$$X_{-1} = f(R) \tag{4}$$

where $f(R)$ is a fixed function of R, the run number of the run under consideration. ($R = R_1 + j$ in the $(j + 1)$th run). The meanings of ψ, ϕ, and π will be stated in a moment.

The lattice structure allows us to have up to 12 bonds leading from each atom; and for the sake of exposition it is convenient to suppose in the first place that each atom has a full complement of 12 bonds from it. The atom A_n thus requires 12 η-values for its 12 bonds: we denote these by $(\eta_n^{(1)}, \eta_n^{(2)}, \cdots, \eta_n^{(12)})$. The quantity ϕ consists of 12 independent bits $\phi_n = (\phi_n^{(1)}, \phi_n^{(2)}, \cdots, \phi_n^{(12)})$ also corresponding to the 12 bonds. If an atom has fewer than 12 bonds from it, we put the corresponding $\phi^{(i)} = 0$ to signalize the absence of a bond. For a bond actually present, we also put $\phi^{(i)} = 0$ if it is *known* that this bond leads to some atom already in the wet list. In all other cases, $\phi^{(i)} = 1$. Thus all outgoing bonds from A_n have $\phi_n^{(i)} = 1$: the converse is not necessarily true, since we may have a bond, which exists and which leads to an atom already in the wet list (and is therefore not an outgoing bond), although at the current stage of the calculation we have not yet discovered that this bond leads to an atom in the wet list. Thus the quantity ϕ represents a state of current knowledge. We define

$$\psi_n = \max\,(\eta_n^{(1)}\phi_n^{(1)}, \cdots, \eta_n^{(12)}\phi_n^{(12)}). \tag{5}$$

To validate this definition we require that each $\eta_n^{(i)}$ be a 9-bit number. We achieve this by means of

$$\left.\begin{aligned}
(X_n) &\equiv (\eta_n^{(1)}, \eta_n^{(2)}, \eta_n^{(3)}, ?) = [9, 9, 9, 9,] \\
(\beta X_n) &\equiv (\eta_n^{(4)}, \eta_n^{(5)}, \eta_n^{(6)}, ?) = [9, 9, 9, 9,] \\
(\beta^2 X_n) &\equiv (\eta_n^{(7)}, \eta_n^{(8)}, \eta_n^{(9)}, ?) = [9, 9, 9, 9,] \\
(\beta^3 X_n) &\equiv (\eta_n^{(10)}, \eta_n^{(11)}, \eta_n^{(12)}, ?) = [9, 9, 9, 9,]
\end{aligned}\right\} (\mathrm{mod} 2^{36}), \tag{6}$$

where ? denotes a 9-bit number which is not used (because the terminal

digits of pseudo-random numbers are too regularly distributed), and where the congruential notation in (6) indicates low multiplication as in (3).

VII. PROGRESSIVE CONSTRUCTION OF THE WET LIST

We are now able to describe recursively how the wet list is compiled. In what follows, it is important to remember that the wet list consists of both source atoms as well as recruited atoms in this list.

Suppose that the wet list is already partly compiled and that we have reached a stage at which the current c value has just been reduced to a new value. Starting at the beginning of the wet list, we successively scan each atom in the order of the list. For each atom scanned we ask first if its ψ-value is less than c. If $\psi < c$, we pass to the next atom on the list. If $\psi \geqq c$, we determine all values of i such that $\eta^{(i)}\phi^{(i)} \geqq c$: these represent the only bonds which can lead to an atom at present qualified for membership of the wet list. Call such an atom a target atom of the scanned atom. Noting the cell coordinates and the type number of the scanned atom, we compute the cell coordinates and the type number of each target atom of the scanned atom. We then look through the wet list to see which of the target atoms do not belong to the wet list, and we add to the end of the wet list all target atoms not already on the wet list. With these new additions to the wet list, every bond from the scanned atom to a target atom leads to an atom of the wet list, and therefore we now set $\phi^{(i)} = 0$ for *all* values of i for which we had $\eta^{(i)}\phi^{(i)} \geqq c$. Next we recompute ψ from (5). Of course the new value of ψ is less than c. Therefore in the scanning procedure we go to the next atom in the wet list. Ultimately the scanning procedure will reach the end of the wet list. At this stage, all ψ in the wet list are less than c. We therefore reduce c to the largest ψ in the wet list, and restart the scanning from the beginning of the wet list. We continue this procedure until we have recruited $N + 1$ atoms to the wet list.

To assist in computing the new value of c required at any reduction of c, we make a small modification of the foregoing procedure. We define a number c^*, called the c-candidate. At the beginning of any scan, we set $c^* = 0$. Before leaving any scanned atom and proceeding to the next one, we take max (ψ, c^*) to be the new value of c^*. Thus when we reach the end of the scan, c^* equals the required new value of c.

VIII. WET LIST SEARCH

As described above, we have to search through the wet list to decide if a target atom is already in the list. To expedite this search we define

the modular type of an atom A_n in the wet list to be the least non-negative residue

$$t_n = x_n + y_n + z_n \pmod{256}, \tag{7}$$

and we dissect the wet list into 256 equivalence classes according to their modular type. A target atom can only be in the wet list if it is its own equivalence class, and hence it is sufficient to search just one equivalence class in the wet list.

Suppose that $A_{n(1)}$, $A_{n(2)}$, $\cdots$, $A_{n(k)}$ are the atoms in the wet list currently belonging to the equivalence class to be searched. We define π_n, appearing in (1), by

$$\pi_{n(j)} = n(j - 1). \tag{8}$$

Thus we can search the equivalence class backwards; for at any stage of the search, the π-value for the atom currently examined gives us the address of the next atom to be examined. To start off this iteration, the core storage holds a table, called the H-table, with 256 entries providing the values of $n(k)$ for each equivalence class. When a new atom has to be added to the wet list, we must write up its π-value: its π-value is simply the entry (for the appropriate equivalence class) that is in the H-table immediately before adding the new atom to the wet list; and, immediately after adding this new atom to the wet list into address $n(k + 1)$, say, we enter $n(k + 1)$ into the appropriate position of the H-table in place of $n(k)$.

IX. INHIBITION OF RUNS

Each of the coordinates x, y, z of a lattice cell is represented by an 11-bit integer, treated by the machine modulo 2048. Thus, effectively, the lattice lies on a four-dimensional torus instead of the required three-dimensional flat of four-dimensional Euclidean space. To remedy this defect, we cut the torus on each of the three two-dimensional flats defined by $x = 0$, $y = 0$, and $z = 0$ respectively. We place the source atoms in, or in the immediate neighborhood of, the cell (1024, 1024, 1024); and we set an inhibition flag if any cell coordinate becomes zero modulo 2048.

The run is allowed to proceed as before, after the inhibitor flag has been set, up to such time as the machine calls for a new value of c. At this instant, however, the inhibitor flag prevents the new value of c being set, and instead terminates the run with the printout comment c *INHIBITED* together with a point of the *old* value of c.

The net effect of this procedure is to allow the fluid to pass across the

two-dimensional cuts in the torus, and even to complete circuits which are not homotopic to zero. However, if such a circuit occurs, it may imply an unnecessary reduction in c, or an unnecessarily large reduction; and in that event the run must be terminated as a safety measure. Nevertheless it is very unlikely, with the values of N used, that inhibition will be invoked; and in fact it has not been invoked on any run calculated to date.

X. GENERATION OF PSEUDO-RANDOM NUMBERS

There is no point in calculating pseudo-random numbers which are not going to be used. At the start of a run, all entries X, defined in (1), are set with a negative sign. When the machine comes to scan any atom in the wet list to look for possible outgoing bonds, it first asks if X is negative. If X is negative, it replaces X by a positive pseudo-random number X_n, generated there and then by means of (3). If X is positive, the machine knows that a pseudo-random number has already been calculated for this atom, and it does not change X_n. Thus, as the problem requires, each individual pseudo-random number remains fixed throughout a run.

XI. DELTA AND PHI TABLES

Consider the stage of the calculation when the machine is scanning the wet list and looking for the possible outgoing bonds from a particular atom A to the corresponding target atoms. Let (x, y, z, T) denote the cell coordinates and type number of A; and suppose that we are considering the ith bond from A $(i = 1, 2, \cdots, 12)$ as a possible outgoing bond to a target atom, whose cell coordinates and type number we denote by (x', y', z', T'). Then

$$E' - E = (x', y', z', T') - (x, y, z, T) = \Delta(T, i) \tag{9}$$

is a function of T and i only. We store $\Delta(T, i)$ in relative location $16T + i$ of a block of 128 locations called the delta table. We can thus calculate the coordinates of the target atom by entering the delta table and using the addition

$$E' = E + \Delta(T, i). \tag{10}$$

Similarly, the value of ϕ a target atom is a function $\phi(T, i)$, stored in relative location $16T + i$ of a block of 128 locations called the phi table; and this enables us to write ϕ' into $E' + 2$ by a straight look-up procedure.

The contents of the delta and phi tables are permanent settings computed from the lattice structure data cards before starting the first run.

REFERENCES

1. Broadbent, S. R., and Hammersley, J. M., Percolation Processes, Crystals and Mazes. Proc. Camb. Phil. Soc., **53**, 1957, p. 629.
2. Vyssotsky, V. A., Gordon, S. B., Frisch, H. L., and Hammersley, J. M., Phys. Rev. **123**, 1961, p. 1566; Frisch, H. L., Sonnenblick, E., Vyssotsky, V. A., and Hammersley, J. M., Phys. Rev. **124**, 1961, p. 1021; Frisch, H. L., Hammersley, J. M., and Welsh, D. J. A., Phys. Rev., to be published.
3. Hammersley, J. M., Monte Carlo Solution of Percolation in the Cubic Crystal, *Computational Methods in the Physical Sciences* (book), to be published.